Programmation I
-
Lisibilité et notion d'algorithme

Jérôme Kirman

Année 2012-2013

Lisibilité

Un programme bien construit possède deux qualités :

- Il doit être fonctionnel (fournir rapidement des résultats corrects).
- Il doit être lisible (compréhensible pour un programmeur).

La lisibilité permet à un lecteur (y compris le concepteur) de comprendre facilement l'objectif et le fonctionnement détaillé du programme ; en lisant directement son code source. Elle implique :

- De choisir des noms explicites.
- D'ajouter des commentaires explicatifs.
- D'indenter/espacer correctement.

Nommage

Le nom d'une variable (ou d'une fonction) doit apporter une information **pertinente** et **concise** sur son contenu (ou sa fonction).

Il existe plusieurs conventions de nommage :

- n
- name
- name_with_underscores
- mixedCaseName
- CamelCaseName
- ALL_CAPS

Certaines sont attribués à un rôle par convention : noms abrégés pour des variables très locales (petite boucle ou courte fonction), majuscules pour les constantes (valeurs fixes permanentes décidées par le programmeur).

Les autres styles sont affaire de choix : seule la cohérence compte.

Commentaires

Tout ce qui se trouve après # sur une ligne est ignoré par l'interpréteur : c'est un *commentaire*.

Les commentaires sont destinés aux lecteurs humains : développeur, collègues, relecteurs, clients, utilisateurs, etc...

Idéalement, ils doivent apporter au lecteur humain des informations sur ce que fait le programme qui ne sont pas évidentes à la lecture de celui-ci. (test de boucle obscur, résumé sémantique d'une fonction, etc...)

Equilibrer les commentaires est délicat, mais facilite le travail ultérieur sur un programme.

Modularité

Un programme complet accomplit une tâche, mais se décompose en de nombreux petits sous-programmes (fonctions, blocs, etc...) dédiés chacun à une sous-tâche élémentaire.

Le code source du programme doit refléter ce décpoupage pour faciliter la lisibilité : chaque tâche élémentaire est accomplie par une fonction portant un nom approprié.

Lorsqu'une fonction accomplit plusieurs tâches on peut soit :

- La décomposer en plusieurs sous-fonctions (si elle est trop longue).
- Marquer un saut de ligne entre chaque groupe logique (espacement).

Indentation

Rappel : l'indentation est le nom donné à l'espace laissé en début de ligne qui permet de déterminer à quel bloc appartient une instruction.

Le type d'indentation (espaces ou tabulations) utilisé en Python est laissé au choix du programmeur, mais il faut rester cohérent. (On trouve couramment 1/2/4/8 espaces, ou 1 tab ; Python recommande 4 espaces.)

Attention au choix d'indentation sur les travaux collectifs : la taille d'une tabulation peut varier (2, 4 ou 8 colonnes, ou plus rarement d'autres valeurs) et chaque éditeur a sa configuration propre (paramétrable).

Affichage de l'indentation

Si vous utilisez l'éditeur Geany, il existe une option pour afficher les espaces/tabulations en début de ligne. La plupart des éditeurs ont une commande équivalente.

Algorithmique et complexité

L'algorithmique est la pratique consistant à produire des algorithmes, c'est-à-dire des suites précises d'instructions visant à trouver la réponse à un problème en général.

Cette notion est indissociable de celle de programmation. Ecrire un programme qui répond à une question suppose de concevoir un algorithme qui calcule le résultat de ce programme.

Deux algorithmes qui trouvent toujours la même réponse ne sont pas forcément identiques : ils peuvent être plus ou moins efficaces.

Un programme correct donne la bonne réponse. Un bon programme donne la bonne réponse en un minimum de temps !

Complexité : exemple

Exemple : vérification orthographique

On a un mot de n lettres et un dictionnaire de m mots. On veut savoir si le mot est bien ortographié.

Un algorithme naïf pourrait se décrire ainsi : "Deux mots sont identiques si toutes leurs lettres sont identiques. On compare chaque mot du dictionnaire au mot à vérifier et, si aucun n'est identique, alors le mot est rejeté. Sinon, il est accepté.".
Cet algorithme est juste (il trouve la bonne réponse), mais inefficace (il va comparer environ m fois n lettres).

Une recherche dans un dictionnaire doit être bien plus rapide : les mots sont rangés par ordre alphabétique. Un algorithme plus intelligent saura (comme un humain) chercher directement au bon endroit, et trouvera la réponse en (environ) n comparaisons. (Et même moins.)

Structures de données

Pour retrouver une information enregistrée plus rapidement (et donc écrire des algorithmes plus efficaces), il faut utiliser les bonnes structures de données.

Lorsqu'une variable contient de nombreuses informations, trouver, ajouter ou supprimer un élément à l'intérieur peut prendre un certain temps.

Il existe de nombreux moyens d'enregistrer des informations, chacun ayant des performances propres pour diverses opérations. L'étude des diverses structures de données utilisées en informatique pourrait faire l'objet d'un cours à part entière, mais il y a trois types de mécanismes essentiels :

- Les tableaux (lecture rapide pour un élément, écriture lente)
- Les listes (lecture lente pour un élément, écriture rapide)
- Les arbres (presque toujours les plus rapides, mais plus compliqués)